Spring Boot Security

Spring Security 是一个能够基于 Spring 的企业应用系统提供声明式的安全访问控制解决方案的安全框架。
它提供了一组可以在 Spring 应用上下文中配置的 Bean，充分利用了 Spring IoC、DI（控制反转 Inversion of Control，DI:Dependency Injection 依赖注入）和 AOP（面向切面编程）功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控制编写大量重复代码的工作。

- Spring Security 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- 用户名和密码，可以在 application.properties 重新进行配置

```
# security
spring.security.user.name=admin
spring.security.user.password=admin
```

登录认证

配置

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                // .loginPage("/login")
                .permitAll()
                .and()
```

```
            .logout()
                .permitAll()
                .and()
            .csrf()
                .ignoringAntMatchers("/logout");

    }
}
```

- @EnableWebSecurity，开启 Spring Security 权限控制和认证功能。
- antMatchers("/", "/home").permitAll()，配置不用登录可以访问的请求。
- anyRequest().authenticated()，表示其他的请求都必须要有权限认证。
- formLogin()，定制登录信息。
- loginPage("/login")，自定义登录地址，若注释掉则使用默认登录页面。
- logout()，退出功能，Spring Security 自动监控了 /logout。
- ignoringAntMatchers("/logout")，Spring Security 默认启用了同源请求控制，在这里选择忽略退出请求的同源限制。

角色权限

也可以在 Java 代码中配置用户登录名和密码，在上面创建的 SecurityConfig 类中添加方法 configureGlobal()。

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
            .passwordEncoder(new BCryptPasswordEncoder())
            .withUser("user")
                .password(new BCryptPasswordEncoder()
                    .encode("123456")).roles("USER");
}
```

在 Spring Boot 2.x 中配置密码需要指明密码的加密方式。当面配置文件和 SecurityConfig 类中都配置了用户名和密码时，会使用代码中的用户名和密码。添加完上述代码，重启项目后，即可用最新的用户名和密码登录系统。

在上述代码中有这么一段 roles("USER") 指明了用户角色，角色就是 Spring Security 最重要的概念之一，往往通过用户来控制权限比较繁琐，在实际项目中，往往都是将用户关联到角色，给角色赋予一定的权限，通过角色来控制用户访问请求。

为了演示不同角色拥有不同权限，再添加一个管理员 admin 和 角色 ADMIN。

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
            .passwordEncoder(new BCryptPasswordEncoder())
            .withUser("user")
                 .password(new BCryptPasswordEncoder()
                    .encode("123456")).roles("USER")
            .and()
            .withUser("admin")
                 .password(new BCryptPasswordEncoder()
                    .encode("admin")).roles("ADMIN", "USER");
}
```

admin 用户拥有 USER 和 ADMIN 的角色，user 用户拥有 USER 角色，添加 admin.html 页面设置只有 ADMIN 角色的用户才可以访问。

添加后端访问：

```
@RequestMapping("/admin")
public String admin() {
return "admin";
}
```

我们再将上述的 configure() 方法修改如下：

```
@Override
protected void configure(HttpSecurity http) throws Exception {
http.authorizeRequests()
.antMatchers("/resources/**", "/").permitAll()
.antMatchers("/admin/**").hasRole("ADMIN")
.antMatchers("/content/**").access("hasRole('ADMIN') or hasRole('USER')")
.anyRequest().authenticated()
.and()
.formLogin()
// .loginPage("/login")
.permitAll()
.and()
.logout()
.permitAll()
.and()
```

```
.csrf()
.ignoringAntMatchers("/logout");
}
```

重点看这些：

- antMatchers("/resources/** ", "/").permitAll()，地址 "/resources/ **" 和 "/" 所有用户都可访问，permitAll 表示该请求任何人都可以访问；
- antMatchers("/admin/** ").hasRole("ADMIN")，地址 "/admin/**" 开头的请求地址，只有拥有 ADMIN 角色的用户才可以访问；
- antMatchers("/content/** ").access("hasRole('ADMIN') or hasRole('USER')")，地址 "/content/**" 开头的请求地址，可以给角色 ADMIN 或者 USER 的用户来使用；
- antMatchers("/admin/**").hasIpAddress("192.168.11.11")，只有固定 IP 地址的用户可以访问。

更多的权限控制方式参看下表：

方法名	解释
access(String)	Spring EL 表达式结果为 true 时可访问
anonymous()	匿名可访问
denyAll()	用户不可以访问
fullyAuthenticated()	用户完全认证可访问（非 remember me 下自动登录)
hasAnyAuthority(String...)	参数中任意权限的用户可访问
hasAnyRole(String...)	参数中任意角色的用户可访问
hasAuthority(String)	某一权限的用户可访问
hasRole(String)	某一角色的用户可访问
permitAll()	所有用户可访问
rememberMe()	允许通过 remember me 登录的用户访问
authenticated()	用户登录后可访问
hasIpAddress(String)	用户来自参数中的 IP 时可访问

方法级别的安全

上面是通过请求路径来控制权限，也可以在方法上添加注解来限制控制访问权限。

`@PreAuthorize / @PostAuthorize`

Spring 的 @PreAuthorize/@PostAuthorize 注解更适合方法级的安全，也支持 Spring EL 表达式语言，提供了基于表达式的访问控制。

- @PreAuthorize 注解：适合进入方法前的权限验证，@PreAuthorize 可以将登录用户的角色 / 权限参数传到方法中。
- @PostAuthorize 注解：使用并不多，在方法执行后再进行权限验证。

```
@PreAuthorize("hasAuthority('ADMIN')")
@RequestMapping("/admin")
public String admin() {
    return "admin";
}
```

这样只要拥有角色 ADMIN 的用户才可以访问此方法。

- @Secured

此注释是用来定义业务方法的安全配置属性的列表，可以在需要安全 [角色 / 权限等] 的方法上指定 @Secured，并且只有那些角色 / 权限的用户才可以调用该方法。如果有人不具备要求的角色 / 权限但试图调用此方法，将会抛出 AccessDenied 异常。

示例：

```
public interface UserService {

    List<User> findAllUsers();

    @Secured("ADMIN")
    void updateUser(User user);

    @Secured({ "USER", "ADMIN" })
    void deleteUser();
}
```

如此项目中便可根据角色来控制用户拥有不同的权限。为了方便演示，内容中所有用户和角色信息均写死在代码中，在实际项目使用中，会将用户、角色、权限控制等信息存储到数据库中，以更加方便灵活的方式去配置整个项目的权限。